METHOD AND SYSTEM FOR DYNAMICALLY ASSOCIATING TYPE INFORMATION AND CREATING AND PROCESSING META-DATA IN A SERVICE ORIENTED ARCHITECTURE

BACKGROUND

[0001] The present invention relates generally to computer architecture systems and, more particularly, to a method and system for dynamically associating type information and creating and processing meta-data in a service oriented architecture.

[0002] Most service oriented computer network systems use Simple Object Access Protocol (SOAP) as an encoding mechanism, and therefore Extensible Markup Language (XML) as the underlying message format. Normally, messages transferred between the client and the service and both parties follow a message format known to both of them such that they can determine the message type and map it to their type system. This determination is typically based on static information such a well-known schema (e.g., published by standard bodies), some previous agreement on the schema (e.g., published by the service provider in a service description such as WSDL), or using a standard set of application programming interfaces. There are also other cases wherein the XML schema information and types are embedded with the message, and the framework knows how to interpret the message. Such characteristics are acceptable for most of the presently utilized message exchange patterns where both the parties are familiar with one other and the message(s) exchanged therebetween.

[0003] On the other hand, it is also desirable to be able to support a message exchange pattern wherein both client and server are flexible such that messages may be sent without being bound to a previous agreement on the schema. Although "open-content" XML schemas are available by using the XML "any" type definition extension, there is still a semantic problem (i.e., meaning and use of the data) associated with flexible, open-content message data. This problem is driven by the static nature of the data and "a priori" agreements between both parties in the exchange pattern. Accordingly, it would be advantageous to be able to exchange semantic type information dynamically along with the message, but without disturbing the exchange pattern.

SUMMARY

[0004] The foregoing discussed drawbacks and deficiencies of the prior art are overcome or alleviated by a method for dynamically associating type information about extensible messages in a service-oriented architecture. In an exemplary embodiment, the method includes configuring a simple object access protocol (SOAP) message header to include message meta-data and semantics, thereby facilitating a dynamic exchange of semantic and meta-data information for open content message exchange between a sender and a receiver.

[0005] In another aspect, a system for dynamically associating type information about extensible messages in a service-oriented architecture includes a simple object access protocol (SOAP) message header configured to include message meta-data and semantics, thereby facilitating a dynamic exchange of semantic and meta-data information for open content message exchange between a sender and a receiver.

[0006] In another aspect, a method for synthesizing and processing dynamically associated meta-data associated with extensible markup language (XML) messages in service-oriented computer architecture includes generating message meta-data within the header of a simple object access protocol (SOAP) message. The SOAP message is received, and semantic information and the meta-data from the SOAP header are retrieved. The semantic information and the meta-data are associated during processing of the body of the SOAP message.

[0007] In yet another aspect, a system for synthesizing and processing dynamically associated meta-data associated with extensible markup language (XML) messages in service-oriented computer architecture includes a send side framework for generating message meta-data within the header of a simple object access protocol (SOAP) message. A receive side framework receives the SOAP message and processes the semantics in the SOAP header, the receive side framework further configured for retrieving semantic information and the meta-data from the SOAP header, and associating the semantic information and meta-data during processing of the body of the SOAP message.

BRIEF DESCRIPTION OF THE DRAWINGS

[0008] Referring to the exemplary drawings wherein like elements are numbered alike in the several Figures:

[0009] Figure 1(a) is a schematic block diagram illustrating an existing SOAP message exchange pattern between a client and a service in a service-oriented architecture, wherein the data types are known by both client and service;

[0010] Figure 1(b) is a schematic block diagram illustrating an existing flexible SOAP message exchange pattern between a client and a service in a service-oriented architecture, wherein no data types are associated with the message;

[0011] Figure 2 is a block diagram illustrating a SOAP message wherein the header thereof includes message semantics for a flexible XML schema, in accordance with an embodiment of the invention; and

[0012] Figure 3 is a block diagram illustrating a framework for synthesizing and processing dynamically associated meta-data associated with extensible messages in service-oriented architecture, in accordance with a further aspect of the invention.

DETAILED DESCRIPTION

[0013] Disclosed herein is a system and method for dynamically associating type information (e.g., message schema type, location, message description and/or other details including native object type mapping information, etc.) about extensible messages in a service-oriented architecture. Briefly stated, a SOAP header is configured so as to carry the message semantics, thereby allowing both parties to exchange the semantic type information dynamically along with the message without disturbing the message exchange pattern. The header information may include a schema of the 'any' type message, RDF description of the message and/or object system types, or other information that can help the target to understand the data.

[0014] In addition, there is further disclosed a framework to synthesize and process the dynamically associated meta-data (e.g., message schema type, location, message description and/or other semantic details including native object type mapping information, etc.) associated with extensible messages in service-oriented architecture. A send-side (message producer) framework is used to support passing type and other meta-data about the message extensions using SOAP headers, while a receive-side (message consumer) framework processes the semantics in the SOAP header, generates the corresponding semantic/meta-data processors and associates the same with the SOAP and/or XML processors. Furthermore, meta-data processors work, in conjunction with the SOAP and/or XML processors, to validate and map the extensible XML messages to the native object systems and/or native semantic processors.

[0015] As stated previously, in a service oriented architecture, messages (i.e., XML messages) are typically exchanged between the client and the service over a Simple Object Access Protocol (SOAP). Figure 1(a) is a schematic block diagram illustrating an existing SOAP message exchange pattern 10 between a client 12 and a service 14 over a SOAP message channel 16 in a service-oriented architecture. In the example illustrated, the data types (stockQuoteSymbol) are known by both client and service, and thus no type is associated with the message as shown by the XML mapping. In contrast, Figure 1(b) illustrates a flexible SOAP message exchange pattern 20 between client 14 and service 16 in a service-oriented architecture, wherein no data types are associated with the message.

[0016] Thus, some of these SOAP messages are flexible and extensible in nature. Normally, these messages are described using a XML schema for XML message type validation and conformance to the schema type system. In particular, a conventional framework supports a XML message extension scheme using the schema language 'any' construct (using xsd:any (for elements) and xsd:anyAttribute (for attributes) with xsd:##any for any namespace). This typeless data type construct provides the maximum flexibility to a message description by allowing an open content model, and also allows a message to contain any well-formed XML from any namespace. Unfortunately, this flexibility imposes a number of restrictions on the existing message processing frameworks.

[0017] There are a number of conventional approaches that have been implemented to deal with and validate this kind of extensible but un-typed message format (open-content model) using the 'any' type. One way is for the SOAP processors to simply ignore untyped messages and assemble them as a XML fragment, thus and leaving it to the service to handle the semantics. However, the service needs prior knowledge of the message in order to properly process and parse the message. Another approach is for a SOAP processor to utilize a type-mapping system to map a native object to an XML fragment, provided that this mapping is defined earlier. This approach is very much static in nature, since both parties know about the types. Still another approach is for the service and SOAP processors to use well-known schema information to map the XML fragment to the known schema and attempt to validate it. However, this may not work in all instances. Alternatively, a message body may be wrapped with some type information and passed to the processor. Although this approach lets the processor know about the details of how to interpret the type and semantics of the message, it is specific to the encoding of the message and thus not flexible for an open content model.

[0018] Therefore, in accordance with an embodiment of the invention, there is disclosed a dynamic and flexible method for providing the message type information for any extensible message fragments without changing the message format and without imposing any "a priori" knowledge on the message. A mechanism is defined to enhance the existing message exchange pattern by allowing the parties in the conversation to send runtime semantics (e.g., message schema type, description and other semantic information including type information) along with the messages rather than relying on the static information on the messages. This dynamic message type association, along with the support of an open-content model of the message, helps to define flexible and extensible message exchange pattern in a service-oriented architecture.

[0019] Accordingly, Figure 2 is a block diagram illustrating a SOAP message 100 including a message header 102 and message body 104, wherein the header 102 includes message semantics information for a flexible XML schema, such as the 'any' data type. The header 102 also preferably contains a pointer to the <any> data that is carried within message body 104. The message body 104, in turn, includes the normal remote procedure call (RPC) and message parameters, as well as a namespace-qualified 'any' type XML fragment. As such, the present invention embodiments support a message exchange pattern where both client and server have the flexibility to send messages without being bound to a previous agreement, but that also overcomes the type safety and rigidity associated with an open content model.

[0020] An example of a SOAP message having the semantic information included in the message header is presented as follows:

Example

• XML Schema definition

[0021] This is a flexible schema for subscribing to a news information service, wherein any information needed for the subscription information can be passed based on the particular need. Therefore, the schema is defined to be flexible with <xsd:any> element information.

```
<xsd:ComplexType name=" SubscribeToNewsInformation">
    <sequence>
         <element name=" subscription Info" type="subscriptionInfoType "/>
</sequence>
</xsd:ComplexType>

<xsd:ComplexType name=" subscription InfoType">
   <xsd:sequence>
         <xsd:any namespace="##any"/>
   </xsd:sequence>
</xsd:ComplexType>
```

• SOAP Message

[0022] As described above, the schema is flexible and open ended. Thus, the client can submit desired subscription information to the server. In addition, the client can add type information (e.g., the schema type of the message or a native object type mapping information) about the subscription message through the SOAP header. Moreover, the server side processor can infer the type of the message by looking into the header and use that information to process the message. Simple type information can be a schema location about the message, or a java class to handle the message, for example.

```
<?xML version="l .0" encoding="UTF-8"?>
<s: Envelope
                    xmlns:s= http://schemas.xmlsoap.org/soap/envelope/xmlns:xsd="
                    http://www.w3.org/2001/XMLSchema"
xmlns:xsi="
                    http://www.w3.org/2001/XMLschema-instance">
<s: Header>
          <m:anyTypeSemantic
                    xmlns:m =" http://www.ibm.com/2002/AnyProcessor"

                    <m :anyTypePointer m reference=" urn:ibm..any.local.1">
                              <m:schemaLocation
                                        value ="http://www.ibm.com/grid/mySchema.xsd" />
                    </m:anyTypePointer>

                    <m:anyTypePointer m: reference=" urn:ibm..any.local.2">
                              <rdf:RDF
                              xmlns:rdf =http//www.w3c.org/199/02/22-rdf-syntax-ns#>
                              </rdf: RDF>
                    </m:anyTypePointer>
          </m :anyTypeSemantic>
</s:Header>

<s: Body>
      <subscribeToNews xmlns=" http://ibm.com/test/subscribe ">
          <nsl: subscribeToNewsInformation
                              xmlns: nsl =" http://ibm.com/test/types">
                    <nsl:subscription Info>
                    <ns4:any
                              xmlns:ns4="urn:ibm..any.local.1">
                              <ns5:address xmlns:ns5="http://ibm .addrbook">
                                        <name>Joshy </name>
                                        <street> Cherry Hill</street>
                                        <zip> 12603</zip>
                              </ns5 :address>
                    </ns4:any>
          </nsl:subscriptionExpression>

          </nsl:subscribeToNews Information>
```

```
</subscribeToNews>
</s:Body>
</s:Envelope>
```

[0023] As illustrated in the exemplary SOAP message, the message semantics are passed along with the message but without changing the message by including the semantics in the header. This provides the ability to send an 'any' message and extend an 'any' message, wherein the message exchange pattern and the type of the message does not affect either the sender or receiver side of the service framework. Advantageously, there is no need for any prior agreement on extensible message types and service providers can define an open-content model XML message format wherein dynamic message semantics are exchanged for each message. The message types may be associated to a message exchange pattern at runtime, while programming hooks and policies can implement this association.

[0024] In conjunction with the above-described method for dynamically associating meta-data information about the XML message in a service-oriented architecture, a framework is also disclosed that can incorporate this semantic processing model with an XML and SOAP processor. More specifically, this disclosure provides a message producer and consumer framework to pass and interpret the meta-data about the message extensions defined at runtime. This framework is built on the SOAP message exchange pattern, and uses a SOAP header framework for meta-data information exchange.

[0025] Figure 3 is a block diagram illustrating a framework 300 for synthesizing and processing dynamically associated meta-data (e.g., message schema type, location, message description and/or other semantic details including native object type mapping information, etc.) associated with extensible messages in service-oriented architecture, in accordance with a further aspect of the invention. Generally, the framework 300 includes a send-side (message producer) framework 302 to support passing meta-data information regarding the message extensions using SOAP message headers (as discussed above), and a receive-side (message consumer) framework 304 to process the semantics in the SOAP header and associate it with the SOAP and/or XML processor.

[0026] The send-side framework 302 includes a send-side SOAP handler 306 that is used to create the message meta-data about the message extension element. Again, the handler 306 uses SOAP headers as a placeholder for message meta-data, including Uniform Resource Identifier (URI) references to the SOAP body elements for which the semantics can be applied. A sender 308 may associate the meta-data during the runtime through runtime type and meta-data association about extended XML messages, using application programming interfaces (API). In other words, the message producers can use API to register runtime message type and other meta-data information about the extensible XML message with a SOAP message header, but without changing the existing SOAP message exchange pattern.

[0027] In addition, the meta-data may be associated based on policies defined for extensible XML messages in the message exchange pattern. That is, the send-side SOAP handler 306 can associate certain policies 310 with the message extension framework, wherein the framework (at runtime) can embed the meta-data with the SOAP header based on the policy defined for the extensible XML message.

[0028] Once a SOAP message 312 is received, a receiver 314 may process the SOAP message 312 (containing the meta-data about the extensible message) and retrieve the semantic information from the SOAP header with a receive-side SOAP handler 316. Thereafter, the receive-side handler 316 associates the semantic information with a SOAP processor 318 and/or an XML (semantic) processor 320 at the time of the SOAP message body processing. This flexibility allows the runtime system to attach any meta-data information processor with the extensible XML message processor.

[0029] The receiver 314 may process the meta-data information using one or more framework models. First, the server-side SOAP message processing handler 306 can retrieve the SOAP header information, create the necessary meta-data processors, and associate that with the SOAP message body processors 318 and/or XML processors 320 (simple API for XML (SAX) parser). This flexible framework will enable the SOAP engine to create meta-data processors based on the meta-data information in the SOAP header and associate that with the runtime system. For example, a processor that works with the current SOAP XML message parser can also load the XML schema needed to process the extended XML message from a URI location as specified in the SOAP header.

[0030] Second, the SOAP header XML data processor 320 can be used to directly process the header information. During the processing of the SOAP header XML fragments, the receive side handler 316 can create necessary meta-data processors 320 and associate that with the SOAP message body processors and/or XML processors (SAX parser). This flexible framework will enable the SOAP engine to create meta-data processors based on the meta-data information in the SOAP header and associate that with the runtime system.

[0031] Third, the SOAP message processor 318, on parsing, may generate warning messages upon encountering XML elements or attributes that are not specified by the XML schema. These are extended XML messages. The runtime system can then use the meta-data from the SOAP header to process the XML message. For example, the XML processor can load the XML schema for the extensible message and attach that to the schema processor.

[0032] Finally, framework 300 includes one or more meta-data processors working in conjunction with the SOAP and XML processors 318, 320 to validate and map the extensible XML messages. More specifically, a set of pluggable meta-data processors (denoted generally at 322) are defined to process the meta-data information associated with a message. For example, a schema generator processor 324 is based on the XML schemaLocation attribute and namespace information associated with an extended XML message. These schema generators can work with the object type system to generate corresponding native objects. Also, an RDF processor 326 is used to understand the semantic of the extensible message, while native processors 328 manage the type system and type mapping information.

[0033] As will be appreciated, advantages of the above described framework architecture include the dynamic association of message type and meta-data for an extensible message, as well as the flexibility in the processing of dynamic meta-data in a service oriented architecture. In addition, compatible extensions are provided to existing SOAP based message exchange patterns, thereby enabling service providers to define an open-content model XML message format and provide the service clients the capability to dynamically associate the meta-data for the content and be able to process that content data based on the meta-data associated with data.

[0034] While the invention has been described with reference to a preferred embodiment or embodiments, it will be understood by those skilled in the art that various changes may be made and equivalents may be substituted for elements thereof without departing from the scope of the invention. In addition, many modifications may be made to adapt a particular situation or material to the teachings of the invention without departing from the essential scope thereof. Therefore, it is intended that the invention not be limited to the particular embodiment disclosed as the best mode contemplated for carrying out this invention, but that the invention will include all embodiments falling within the scope of the appended claims.